File cheat sheet

```
create empty file       :  newFile, err := os.Create("test.txt")

truncate a file         :  err := os.Truncate("test.txt", 100)

get file info           :  fileInfo, err := os.Stat("test.txt")

rename a file           :  err := os.Rename(oldPath, newPath)

delete a file           :  err := os.Remove("test.txt")

open a file for reading :  file, err := os.Open("test.txt")

open a file             :  file, err := os.OpenFile("test.txt", os.O_APPEND, 0666)
         file open flags:
                           os.O_RDONLY  -  Read only
                           os.O_WRONLY  -  Write only
                           os.O_RDWR    -  Read and write
                           os.O_APPEND  -  Append to end of file when writing
                           os.O_CREATE  -  Create a new file if none exist
                           os.O_EXCL    -  Used with O_CREATE, file must not exist
                           os.O_TRUNC   -  Truncate file when opening

close a file            :    err := file.Close()

write bytes to file     :    n, err := file.Write([]byte("hello, world!\n"))

write at offset         :    n, err := file.WriteAt([]byte("Hello"), 10)

read to bytes           :    n, err := file.Read(byteSlice)

read from offset        :    n, err := file.ReadAt(byteSlice, 10)
```

Create Empty File

```
package main

import (
      "fmt"
    "os"
)


func main() {
    newFile, err := os.Create("test.txt")
    if err != nil {
        fmt.Println(err)
            return
    }
    newFile.Close()
}
```

Rename a File

```
package main

import (
    "fmt"
    "os"
)

func main() {
    originalName := "test.txt"
    newName := "test2.txt"
    err := os.Rename(originalName, newName)
    if err != nil {
        fmt.Println(err)
        return
    }
}
```

Delete a File

```
package main

import (
    "fmt"
    "os"
)

func main() {
    err := os.Remove("test.txt")
    if err != nil {
        fmt.Println(err)
        return
    }
}
```

Check if File Exists

```
package main

import (
    "fmt"
    "os"
)

func main() {
    // Stat returns file info. It will return
    // an error if there is no file.
    fileInfo, err := os.Stat("test.txt")
      if err != nil  {
        if os.IsNotExist(err) {
            fmt.Println("File does not exist.")
        }
        return
    }
      fmt.Println("File", fileInfo.Name(), "does exist.")
}
```

Open and Close Files

```
package main

import (
    "fmt"
    "os"
)

func main() {
    // Simple read only open.
    file, err := os.Open("test.txt")
    if err != nil {
        fmt.Println(err)
        return
    }
    file.Close()

    // OpenFile with more options. Last param is the permission mode
    // Second param is the attributes when opening
    file, err = os.OpenFile("test.txt", os.O_APPEND, 0666)
    if err != nil {
        fmt.Println(err)
        return
    }
    file.Close()

    // Use these attributes individually or combined with an OR for
    // second arg of OpenFile()
    // e.g. os.O_CREATE|os.O_APPEND or os.O_CREATE|os.O_TRUNC|os.O_WRONLY

    // os.O_RDONLY // Read only
    // os.O_WRONLY // Write only
    // os.O_RDWR // Read and write
    // os.O_APPEND // Append to end of file
    // os.O_CREATE // Create is none exist
    // os.O_TRUNC // Truncate file when opening
}
```

####Copy a File

```
package main

import (
    "fmt"
    "os"
)

// Copy a file
func main() {
    // Open original file
    originalFile, err := os.Open("test.txt")
    if err != nil {
        fmt.Println(err)
        return
    }
    defer originalFile.Close()

    // Create new file
    newFile, err := os.Create("test_copy.txt")
    if err != nil {
        fmt.Println(err)
        return
    }
    defer newFile.Close()

    // Copy the bytes to destination from source
    info, _ := os.Stat("test.txt")
    buffer := make ([]byte, info.Size())

    _, err = originalFile.Read(buffer)
    if err != nil {
        fmt.Println(err)
        return
    }
    bytesWritten, err := newFile.Write(buffer)
    if err != nil {
        fmt.Println(err)
        return
    }

    fmt.Printf("Copied %d bytes.\n", bytesWritten)

    // Commit the file contents. Flushes memory to disk
    err = newFile.Sync()
    if err != nil {
        fmt.Println(err)
        return
    }
}
```

####Seek Positions in File

```
package main

import (
    "os"
    "fmt"
)

func main() {
    file, err := os.Open("test.txt")
    defer file.Close()
    if err != nil {
        fmt.Println(err)
        return
    }

    // Offset is how many bytes to move. Can be positive or negative
    var offset int64 = 5

    // Whence is the point of reference for offset
    // 0 = Beginning of file
    // 1 = Current position
    // 2 = End of file

    var whence int = 0
    newPosition, err := file.Seek(offset, whence)
    if err != nil {
        fmt.Println(err)
        return
    }
    fmt.Println("Just moved to 5:", newPosition)

    // Go back 2 bytes from current position
    newPosition, err = file.Seek(-2, 1)
    if err != nil {
        fmt.Println(err)
        return
    }
    fmt.Println("Just moved back two:", newPosition)

    // Find the current position by getting the
    // return value from Seek after moving 0 bytes
    currentPosition, err := file.Seek(0, 1)
    fmt.Println("Current position:", currentPosition)

    // Go to beginning of file
    newPosition, err = file.Seek(0, 0)
    if err != nil {
        fmt.Println(err)
        return
    }
    fmt.Println("Position after seeking 0,0 is ", newPosition)
}
```

Truncate a File

```
package main

import (
    "fmt"
    "os"
)

func main() {
    // Truncate a file to 100 bytes. If file is less than 100 bytes
    // the original contents will remain at the beginning, and the
    // rest  of the space is filled will null bytes.
    // Everything past 100 bytes will be lost.
    // Either way we will end up with exactly 100 bytes.
    // Pass in 0 to truncate to a completely empty file

    err := os.Truncate("test.txt", 100)
    if err != nil {
        fmt.Println(err)
        return
    }
}
```

Get File Info

```
package main

import (
    "fmt"
    "os"
)

func main() {
    // Stat returns file info without file opening.
    //  It will return an error if there is no file.
    fileInfo, err := os.Stat("test.txt")
    if err != nil {
        fmt.Println(err)
        return
    }
    fmt.Println("File name:", fileInfo.Name())
    fmt.Println("Size in bytes:", fileInfo.Size())
    fmt.Println("Last modified:", fileInfo.ModTime())
    fmt.Println("Is Directory: ", fileInfo.IsDir())
}

/*
File name: test.txt
Size in bytes: 100
Last modified: 2019-02-27 12:16:14 +0200 EET
Is Directory:  false
*/
```

File info modification. Time manipulations.

```
package main

import (
      "os"
      "fmt"
      "time"
)

func main() {
      finfo, _ := os.Stat(os.Args[0])
      fmt.Println(finfo.Size(), finfo.ModTime())                // 1

      os.Mkdir("test.dir", 0777)
      finfo, _ = os.Stat("test.dir")
      fmt.Println(finfo.Size(), finfo.ModTime())                // 2

      newFile, err := os.Create("test.txt")
      if err != nil {
            fmt.Println(err)
            return
      }
      newFile.Close()
      finfo, _ = os.Stat("test.txt")
      fmt.Println(finfo.Size(), finfo.ModTime())                // 3
      mtime := time.Date(2006, time.February, 1, 21, 42, 5, 0, time.UTC)
      atime := time.Date(2007, time.March, 8, 14, 15, 0, 0, time.UTC)
      if err := os.Chtimes("test.txt", atime, mtime); err != nil {
            fmt.Println(err)
            os.Exit(1)
      }
      finfo, _ = os.Stat("test.txt")
      fmt.Println(finfo.Size(), finfo.ModTime())                // 4

      loc := time.FixedZone("UTC+2", +2*60*60)
      t := time.Date(2009, time.November, 10, 23, 0, 0, 0, loc)
      fmt.Println("The time is:", t)                            // 5
      t = time.Now()
      fmt.Println("The time is:", t)                            // 6
      fmt.Println(t.Date())                                     // 7
      fmt.Println(t.Clock())                                    // 8
      fmt.Println(t.YearDay())                                  // 9
      fmt.Println(t.Before(time.Now()))                         //10
}

/*
1636864 2019-12-29 09:31:42.9262917 +0200 EET
0 2019-12-29 09:26:18 +0200 EET
0 2019-12-29 09:31:44 +0200 EET
0 2006-02-01 23:42:06 +0200 EET
The time is: 2009-11-10 23:00:00 +0200 UTC+2
The time is: 2019-12-29 09:31:44.0017607 +0200 EET m=+0.017988101
2019 December 29
9 31 44
363
false
*/
```